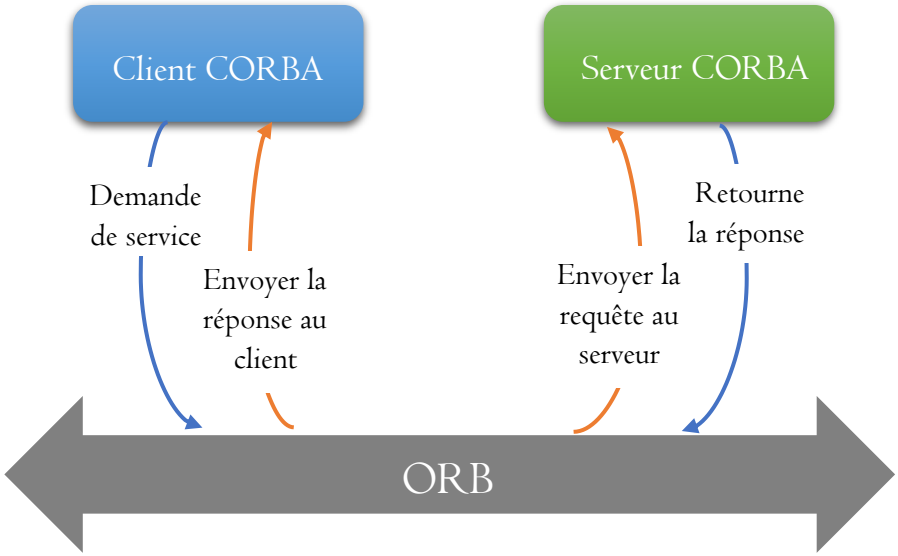
5.I Le modèle de composants CORBA

Common Object Request Broker, plus connu sous l'acronyme CORBA, est une architecture logicielle **à base de composants** qui permet le développement des applications distribuées.

Selon Wikipédia[1], CORBA est défini par l'OMG comme une " architecture logicielle pour **le développement de composants** et d'Object Request Broker (**ORB**). Ces composants, qui sont assemblés afin de construire des applications complètes, peuvent être écrits dans des langages de programmation distincts, être exécutés dans des processus séparés, voire être déployés sur des machines distinctes. "

CORBA est un standard d'interopérabilité bâti sur un modèle client/serveur réparti qui repose sur un **middleware** de communication hétérogène multi-plateforme (matérielle), multi-système et multi-langage.



FigI : Modèle de programmation CORBA

Dans ce qui suit, on va définir les concepts de base qui nous facilite la compréhension des définitions précédentes.

[1] https://fr.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture

L'ORB

ORB qui est l'acronyme de Object Request Broker, est un bus logiciel virtuel qui interconnecte un client avec un serveur dans une architecture client/serveur distribuée. À travers ce bus, le client envoie des requêtes aux serveur pour solliciter les services des objets qu'il héberge.
Grace aux ORBs, la localisation et l'accès aux objets distants se faire d'une façon potable et transparente.
Comme le montre la figure, le client ne communique pas directement avec le serveur, mais avec des composants intermédiaires à savoir le stub et le skeleton. Ces derniers, jouent le rôle d'un proxy qui représente l'image locale d'une une entité distante.

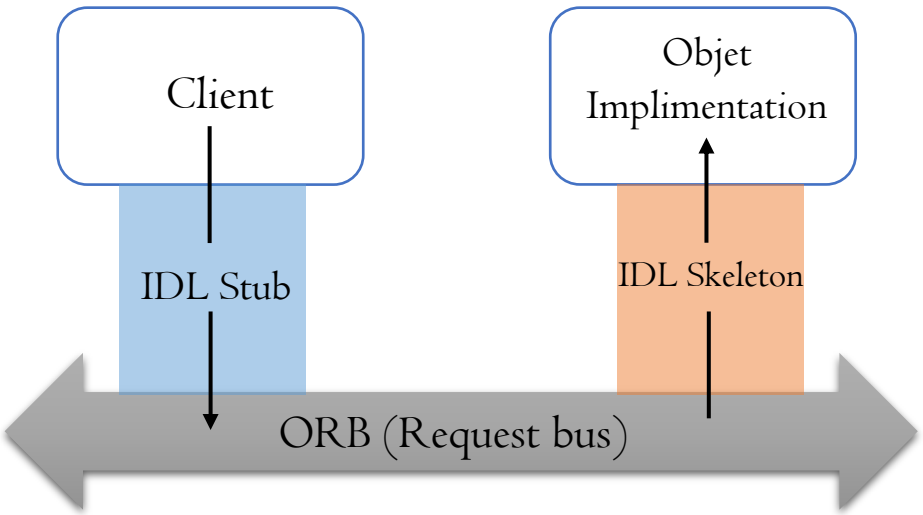


Fig2 : La structure simplifié d'un ORB

Note

Le stub et le skeleton dans le modèle CORBA préservent les mêmes définitions des mêmes entités dans la technologie RMI qu'on a vu précédemment.

Le langage IDL

Interface Definition Language ou Le langage de définition des interfaces est le langage utilisé par CORBA pour la définition des interfaces qui mettre en connexion les composants d'une application répartie.

C'est grâce à ce langage, CORBA assurer l'interopérabilité entre les applications hétérogène en termes de langage de programmation. Par exemple, une application cliente java peut invoquer une méthode d'un objet serveur implémenté en C++.

Pour se souvenir de l'interface, c'est celle qui expose l'ensemble des services (méthodes) offerts par un objet distant.

```
interface Horloge
{
        string donne_heure_a_Msila();
        string donne_heure_a_BBA();
};
```

Fig3 : Exemple d'interface IDL

Après la définition de l'interface, on la compile la compilation pour générer un ensemble d'entités à savoir le stub et le skeleton.

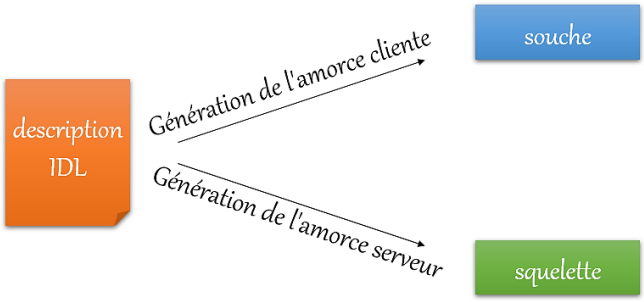
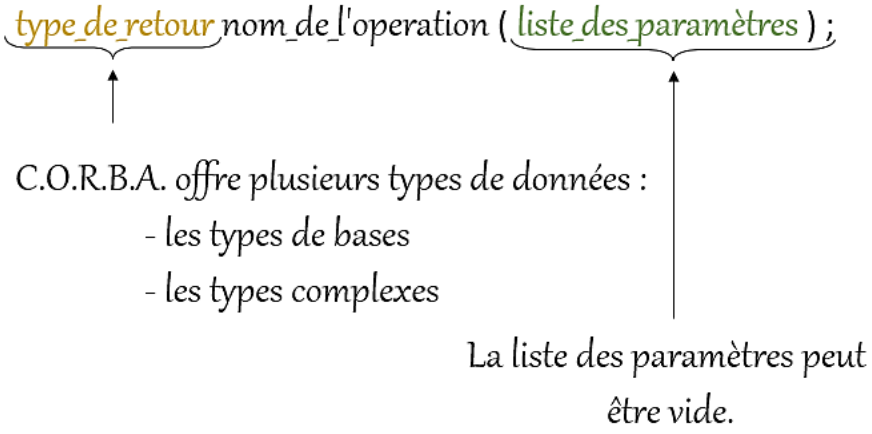


Fig4 : Exemple d'interface IDL

Les caractéristiques d'une interface IDL

Parmi les caractéristiques clé d'une interface IDL, on peut citer :

- Elle possède une extension ".idl".
- Les opérations décrites dans tel interface respectent un format spécial



- Supporte l'héritage multiple.
- La description d'un paramètre comporte trois parties :
  - Le modificateur (in/out)
  - Le type de l'argument (type de base ou type complexe)
  - Le nom de l'argument

Les types de bases de CORBA

Comme tous les langages de programmation, CORBA possède ses propres types de données qui sont utilisés dans la définition des interfaces avec le langage IDL.
Parmi les types de donnée CORBA, on peut citer : le type boolean, octet, short, long, double, char, wchar, string, wstring, etc.

Exemple d'application

On définit dans cet exemple une interface qui comporte une méthode qui se nomme "affiche" et qui affichage un message (passé en paramètres).

```
interface Premier
{
        void affiche ( in string message ) ;
};
```

Cette interface est compilée par la commande :

idlj –fall –v Premier.idl

Après la compilation, plusieurs fichiers sont générés à savoir :

- **PremierPOA.java** : il s'agit du skeleton.
- **_PremierStub.java** : il s'agit du stub
- **Premier.java** : il s'agit de l'interface
- **PremierOperations.java :** il s'agit des opérations de l'interface.

Pour l'exécution, on doit suivre les étapes suivantes par ordre :

- On lancer le serveur
- On met le fichier contenant la référence de l'objet contenant les services à sollicité sur le poste client
- On lance le client.